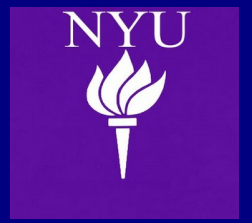
Optimization & Deep Learning

Yann Le Cun
Facebook AI Research,
Center for Data Science, NYU
Courant Institute of Mathematical Sciences, NYU
http://yann.lecun.com



The Convergence of Gradient Descent

$$\omega \leftarrow \omega - \eta \frac{\partial E}{\partial \omega}$$

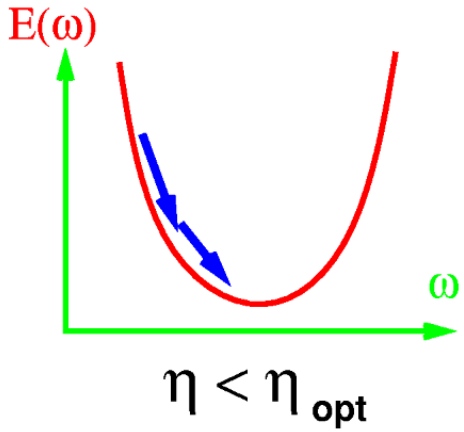
gradient of objective function

weight vector

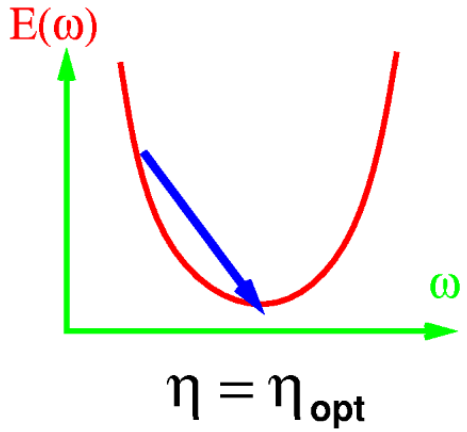
learning rate

- Batch Gradient
- There is an optimal learning rate
- Equal to inverse 2nd derivative

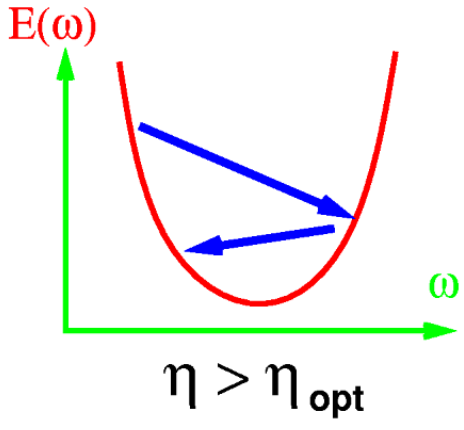
$$\eta_{opt} = \left(\frac{\partial^2 E}{\partial \omega^2} \right)^{-1}$$



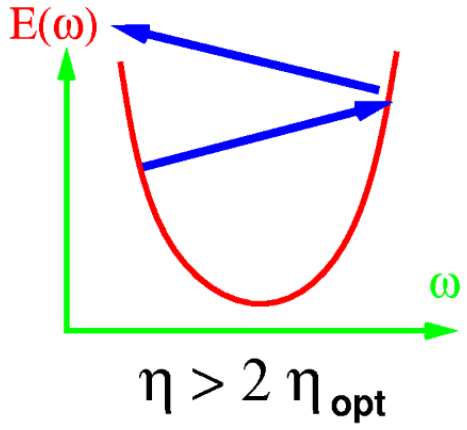
$\eta < \eta_{opt}$



$\eta = \eta_{opt}$



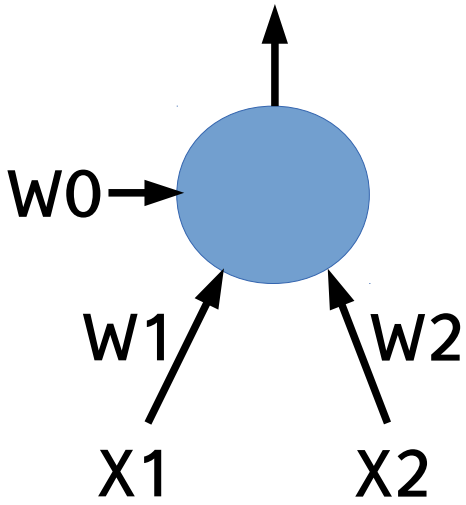
$\eta > \eta_{opt}$



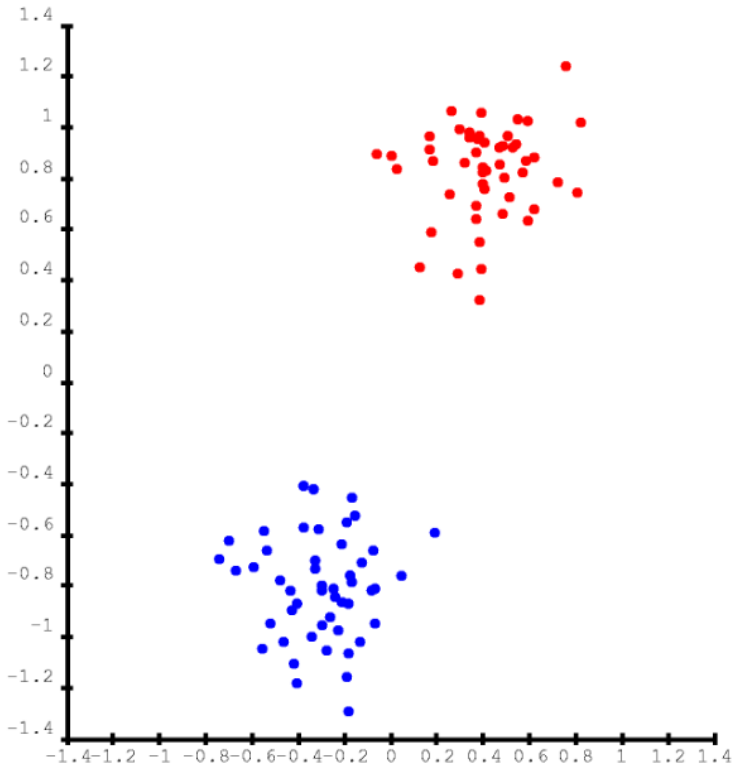
$\eta > 2\,\eta_{opt}$

Let's Look at a single linear unit

- Single unit, 2 inputs



- Quadratic loss
  - $E(W) = 1/p \sum_p (Y - W \cdot X_p)^2$
- Dataset: classification: Y=-1 for blue, +1 for red



- Hessian is covariance matrix of input vectors
  - $H = 1/p \sum X_p X_p^T$
- To avoid ill conditioning: normalize the inputs
  - Zero mean
  - Unit variance for all variable

Convergence is Slow When Hessian has Different Eigenvalues

Y LeCun

Batch Gradient, small learning rate

Learning rate:

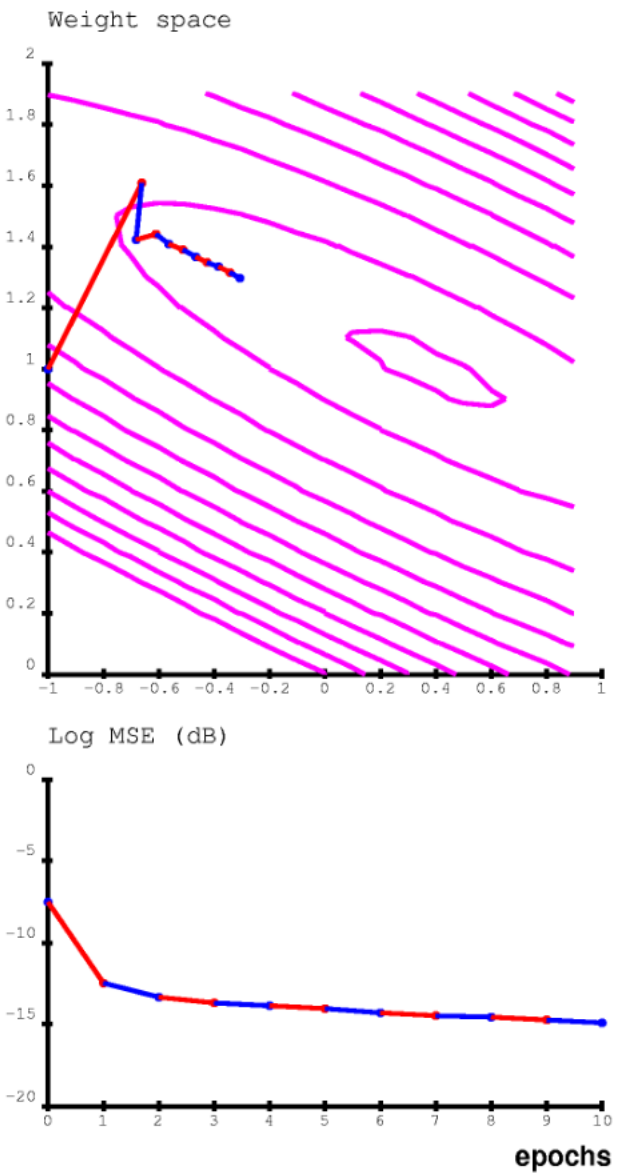
$\eta = 1.5$

Hessian largest eigenvalue:

$\lambda_{max} = 0.84$

Maximum admissible Learning rate:

$\eta_{max} = 2.38$



Batch Gradient, large learning rate

Learning rate:

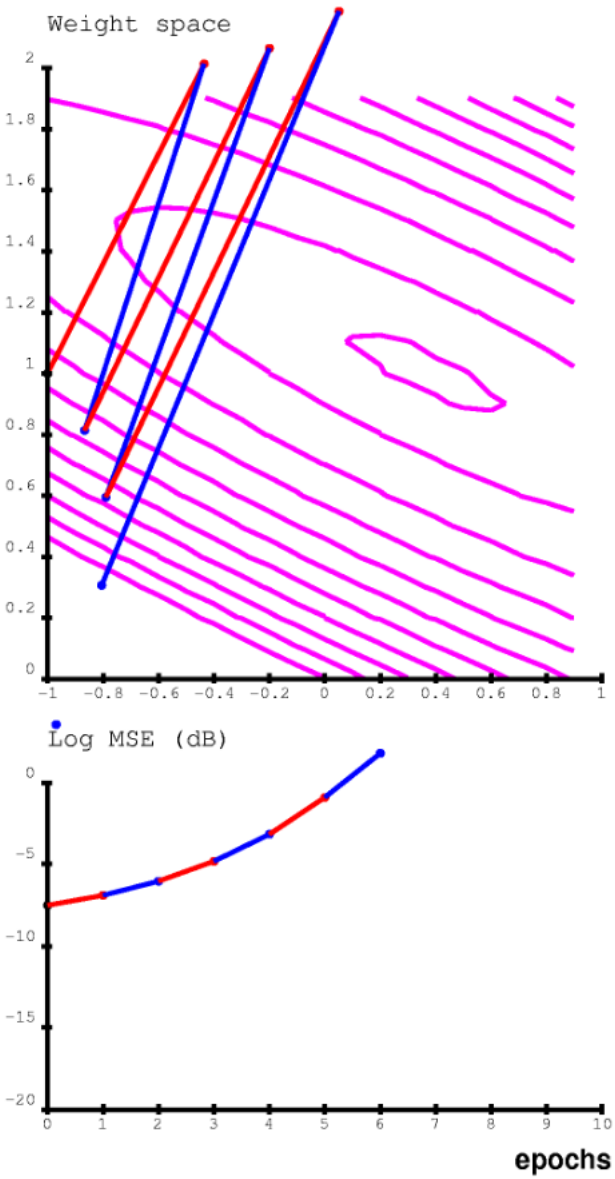
$\eta = 2.5$

Hessian largest eigenvalue:

$\lambda_{max} = 0.84$

Maximum admissible Learning rate:

$\eta_{max} = 2.38$



Convergence is Slow When Hessian has Different Eigenvalues

Y LeCun

- Batch Gradient, small learning rate

Learning rate:

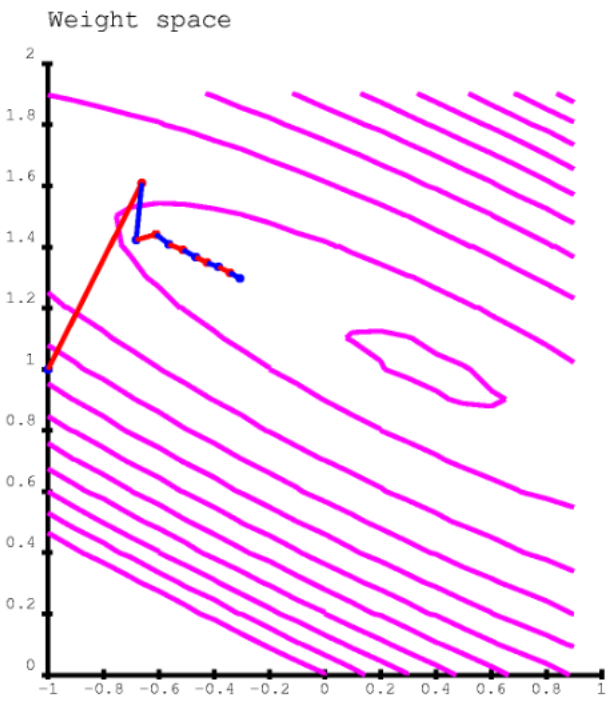
$\eta = 1.5$

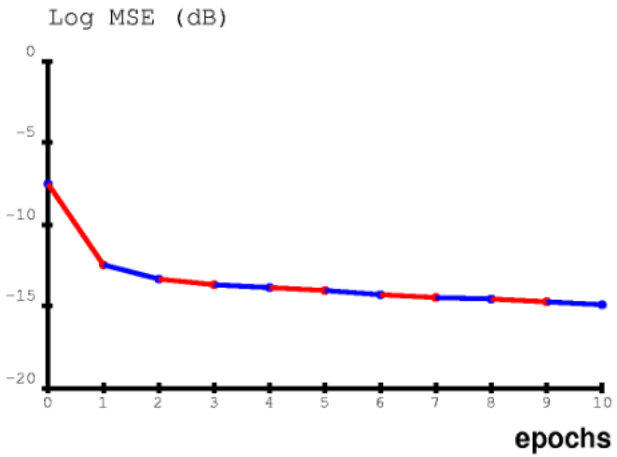
Hessian largest eigenvalue:

$\lambda_{max} = 0.84$

Maximum admissible Learning rate:

$\eta_{max} = 2.38$





- Stochastic Gradient: Much Faster
- But fluctuates near the minimum

Learning rate:

$\eta = 0.2$

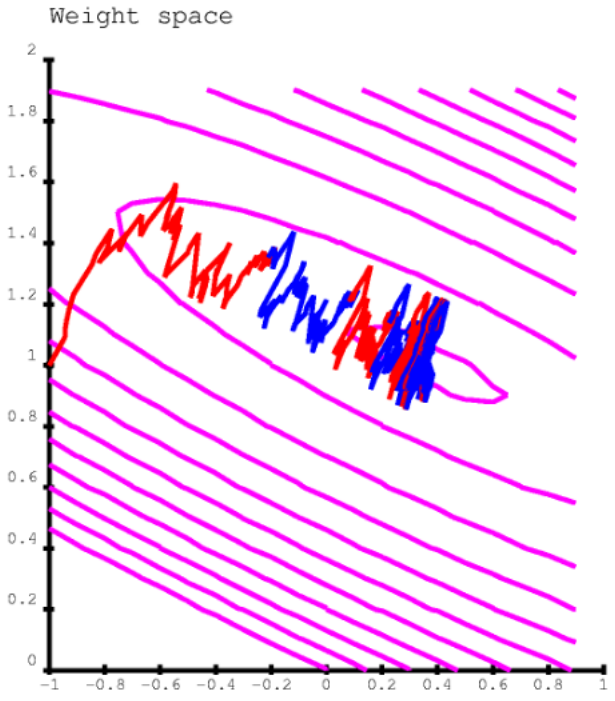
(equivalent to a batch learning rate of 20)

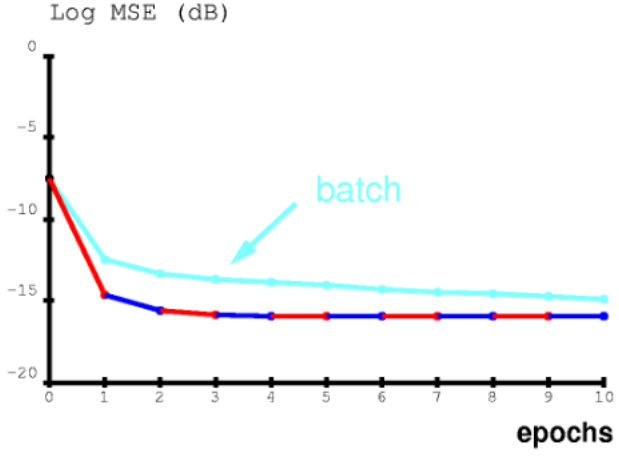
Hessian largest eigenvalue:

$\lambda_{max} = 0.84$

Maximum admissible Learning rate (for batch):

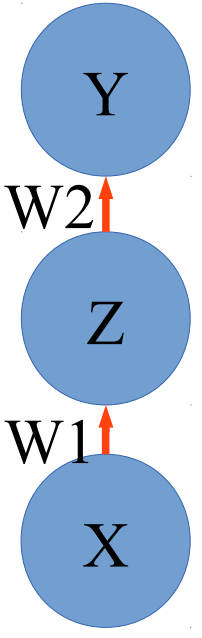
$\eta_{max} = 2.38$

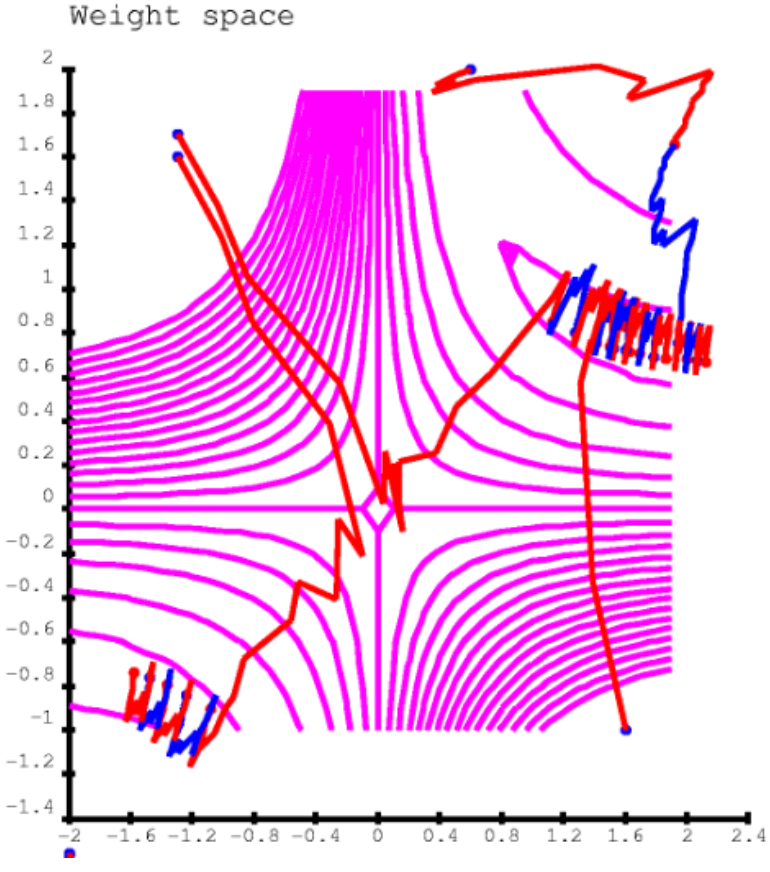


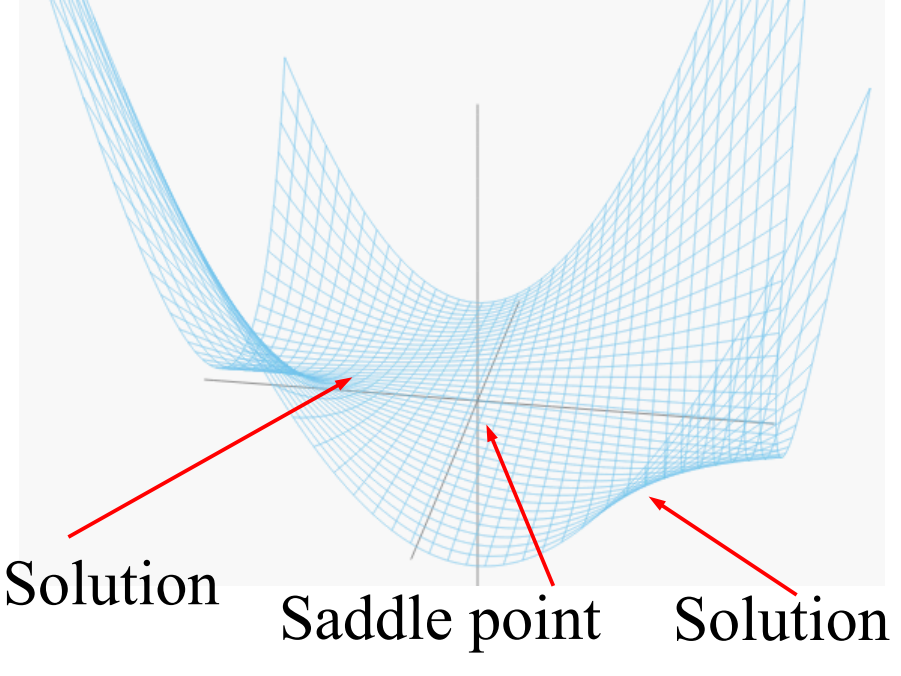


Multilayer Nets Have Non-Convex Objective Functions

- 1-1-1 network
  - Y = W1*W2*X
- trained to compute the identity function with quadratic loss
  - Single sample X=1, Y=1 L(W) = (1-W1*W2)^2
- Solution: W2 = 1/W2 hyperbola.



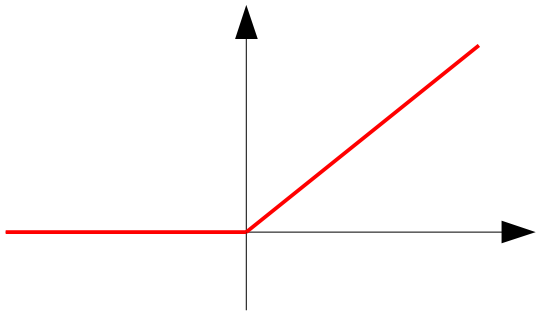


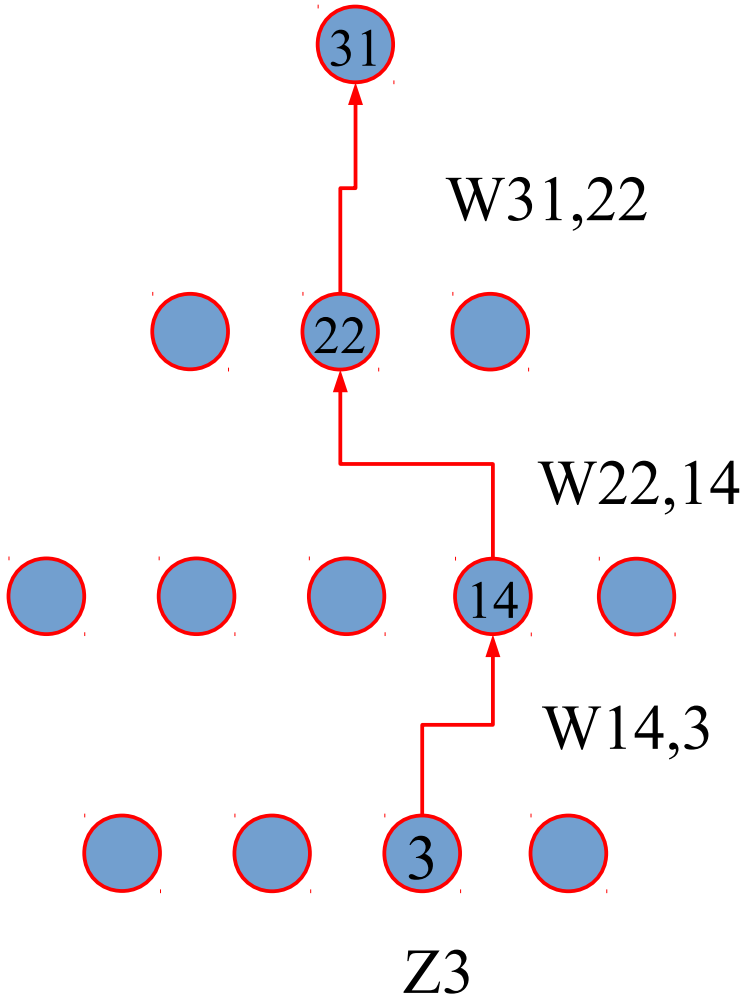


Deep Nets with ReLUs and Max Pooling

- Stack of linear transforms interspersed with Max operators
- Point-wise ReLUs:

$$ReLU(x) = max(x, 0)$$





- Max Pooling
  - "switches" from one layer to the next
- Input-output function
  - Sum over active paths
  - Product of all weights along the path
  - Solutions are hyperbolas
- Objective function is full of saddle points

Geometry of the Loss Function

Deep Nets with ReLUs

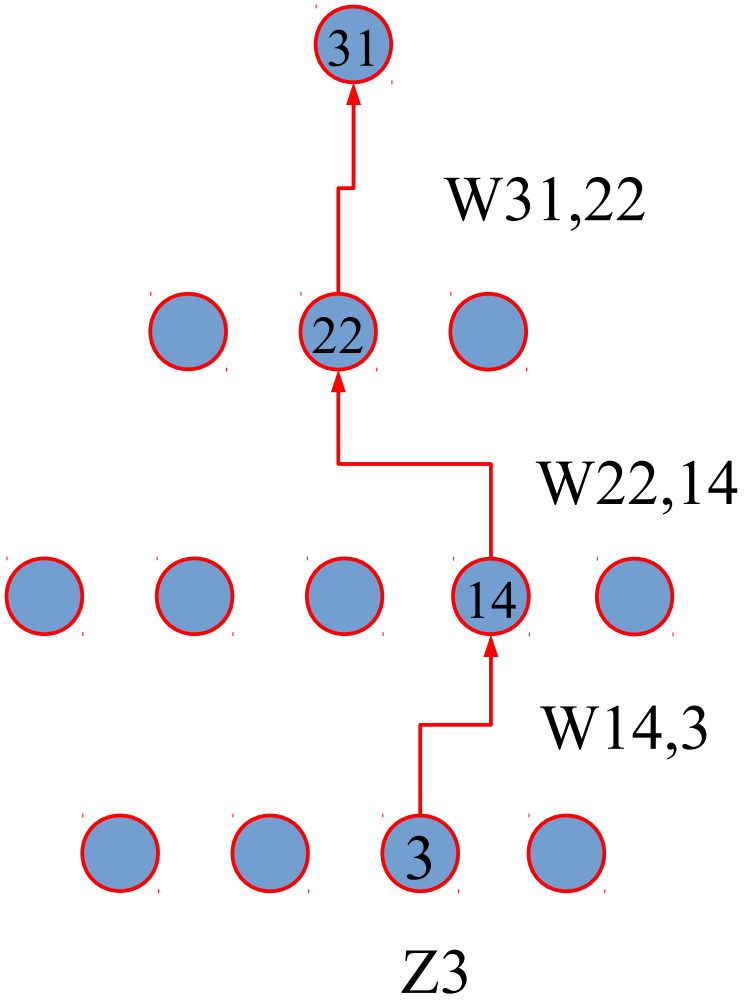
- Single output:

$$\hat{Y} = \sum_P \delta_P(W, X)\left(\prod_{(ij)\in P} W_{ij}\right) X_{P_{start}}$$

- Wij: weight from j to i
- P: path in network from input to output
  - P=(3,(14,3),(22,14),(31,22))
- di: 1 if ReLU i is linear, 0 if saturated.
- Xpstart: input unit for path P.

$$\hat{Y} = \sum_P \delta_P(W, X)\left(\prod_{(ij)\in P} W_{ij}\right) X_{P_{start}}$$

- Dp(W,X): 1 if path P is "active", 0 if inactive
- Input-output function is piece-wise linear
- Polynomial in W with random coefficients



Deep Convolutional Nets (and other deep neural nets)

- Training sample: (Xi,Yi) k=1 to K
- Objective function (with margin-type loss = ReLU)

$$L(W)=\sum_k ReLU\left(1-Y^k\sum_P \delta_P(W,X^k)\left(\prod_{(ij)\in P} W_{ij}\right)X^k_{P_{start}}\right)$$

$$L(W)=\sum_k\sum_P \left(X^k_{P_{start}}Y^k\right)\delta_P(W,X^k)\left(\prod_{(ij)\in P} W_{ij}\right)$$

$$L(W)=\sum_P \left[\sum_k \left(X^k_{P_{start}}Y^k\right)\delta_P(W,X^k)\right]\left(\prod_{(ij)\in P} W_{ij}\right)$$

$$L(W)=\sum_P C_p(X,Y,W)\left(\prod_{(ij)\in P} W_{ij}\right)$$

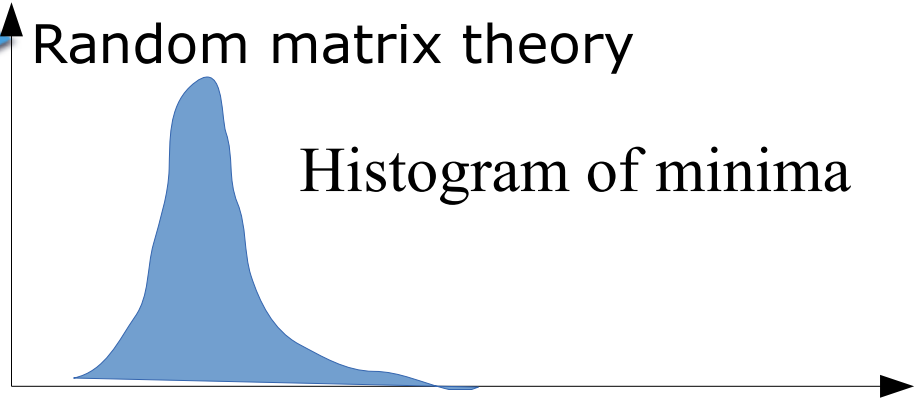
- Polynomial in W of degree l (number of adaptive layers)
- Continuous, piece-wise polynomial with "switched" and partially random coefficients
  - Coefficients are switched in an out depending on W

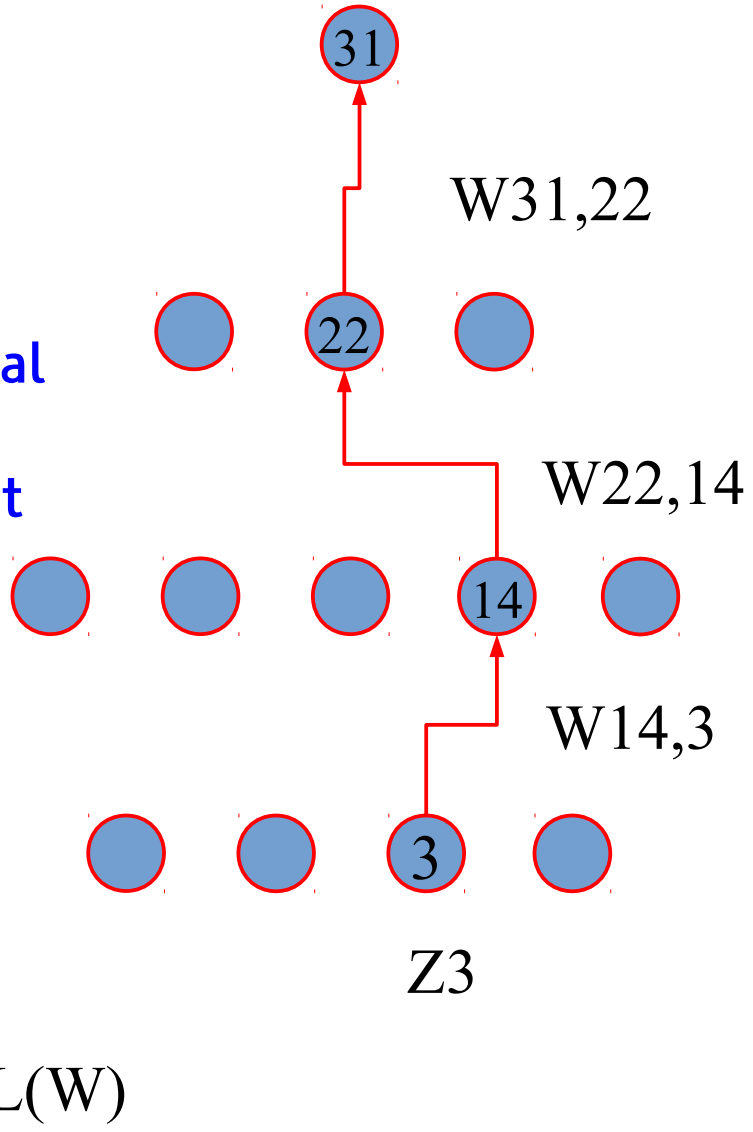
Deep Nets with ReLUs: Objective Function is Piecewise Polynomial

- If we use a hinge loss, delta now depends on label Yk:

$$L(W)=\sum_P C_p(X,Y,W)\left(\prod_{(ij)\in P} W_{ij}\right)$$

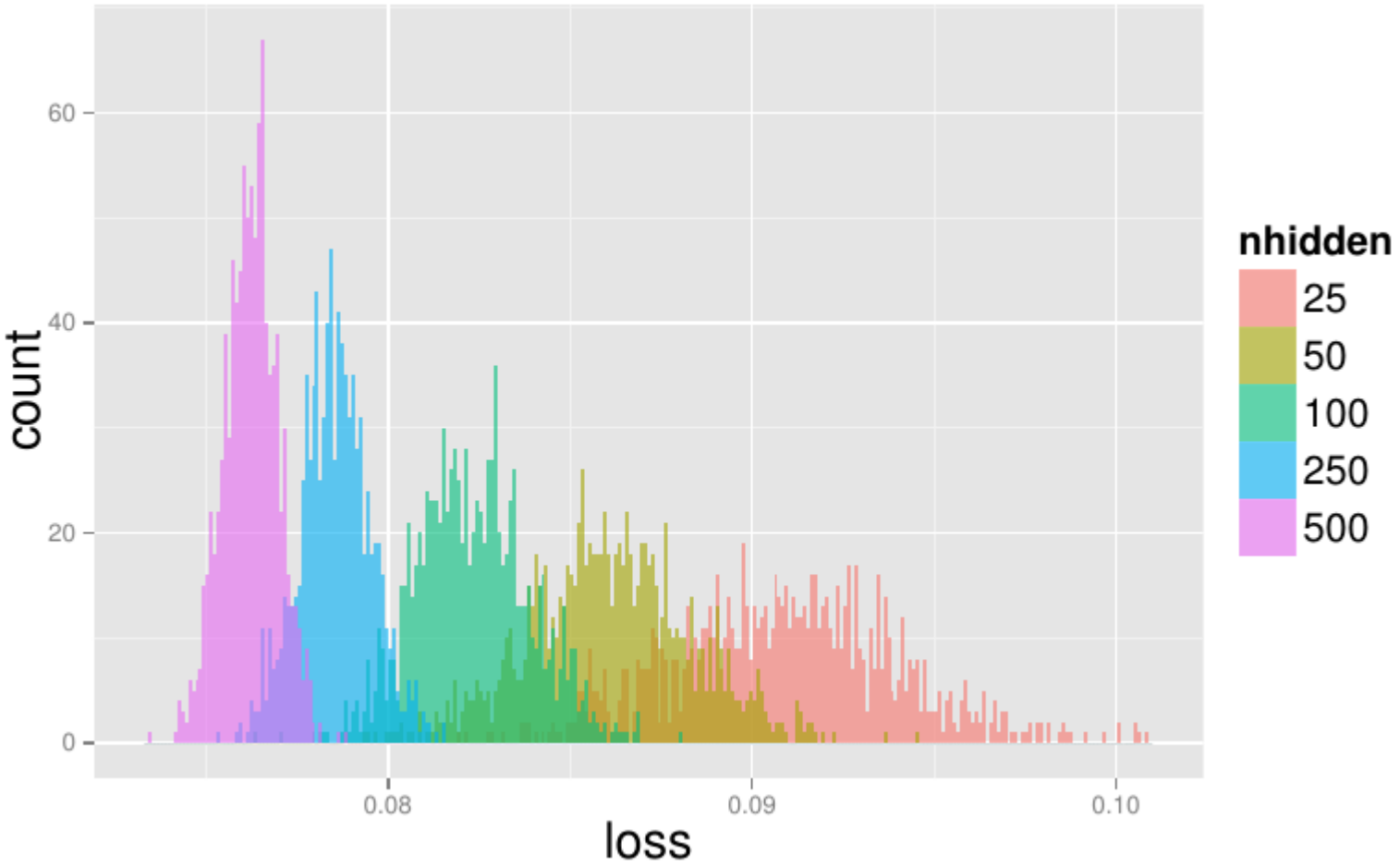
- Piecewise polynomial in W with random coefficients
- A lot is known about the distribution of critical points of polynomials on the sphere with random (Gaussian) coefficients [Ben Arous et al.]
  - High-order spherical spin glasses
  - Random matrix theory





Deep Nets with ReLUs: Objective Function is Piecewise Polynomial

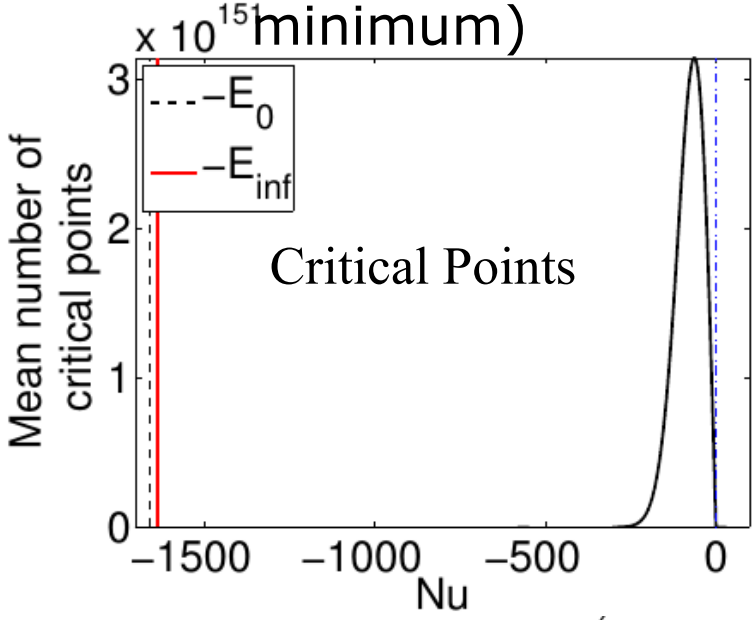
- Train 2-layer nets on scaled-down MNIST (10x10) from multiple initial conditions. Measure loss on test set.

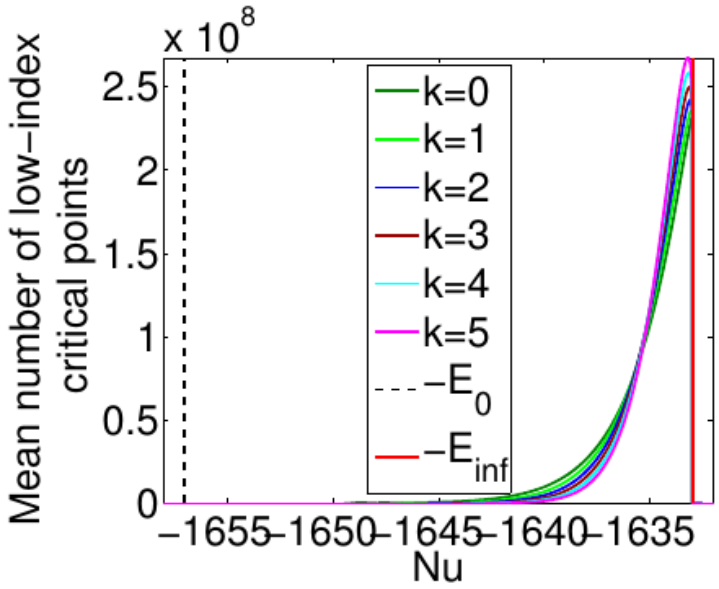


[Choromanska, Henaff, Mathieu, Ben Arous, LeCun 2015]

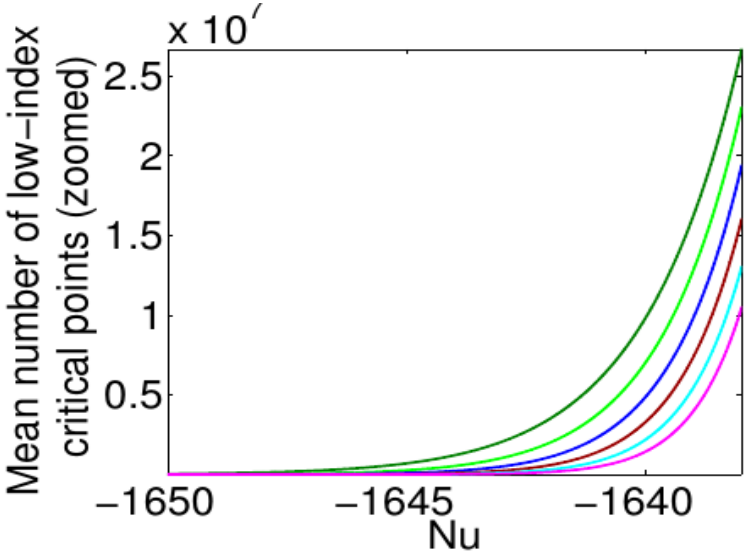
Distribution of critical points (saddle points, minima, maxima)

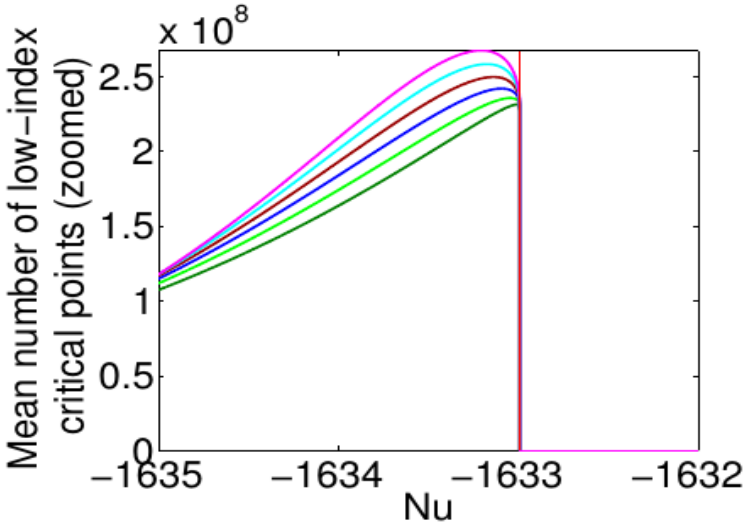
- K=number of negative eigenvalues of Hessian (K=0 → minimum)





Zoomed:






“Target Prop”

Backprop as Optimization under Constraints

- Lagrangian formulation of backrop

$$L(Z,\lambda,W)=D(G_{s-1},Z_s)+\sum_{i=1}^{s}\lambda_i^T[Z_i-G_i(Z_{i-1},W_i)]$$

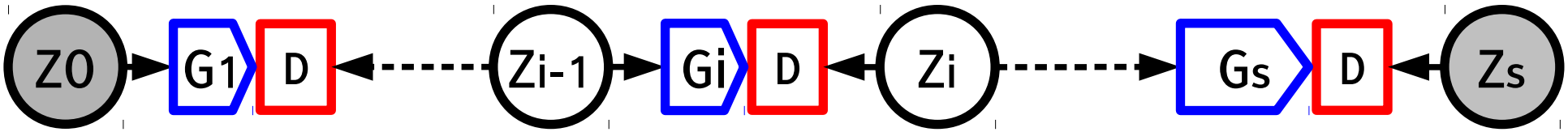
$$\partial L(Z,\lambda,W)/\partial\lambda^i=0 \quad \Rightarrow \quad Z_i=G_i(Z_{i-1},W_i)$$

$$\partial L(Z,\lambda,W)/\partial Z_{i-1}^i=0 \quad \Rightarrow \quad \lambda_{i-1}=[\partial G_i(Z_{i-1},W_i)/\partial Z_{i-1}]\lambda_i$$

- Relax the Constraint: Zi becomes a "virtual target" for layer i

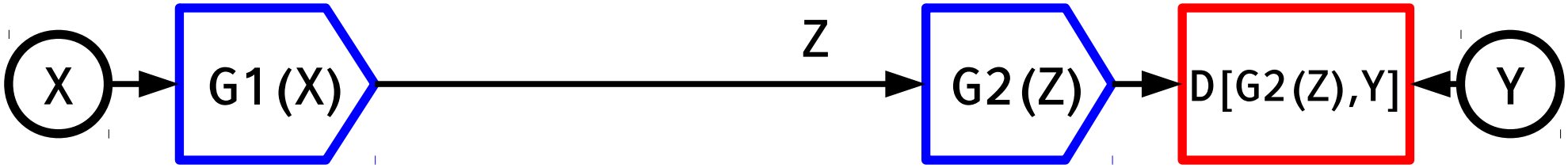
$$L(Z,\lambda,W)=D(G_{s-1},Z_s)+\sum_{i=1}^{s}\alpha_i\|Z_i-G_i(Z_{i-1},W_i)\|^2$$

- Requires an optimization step with respect to virtual targets Z
- Allows to add penalties Z, such as sparsity, quantization....

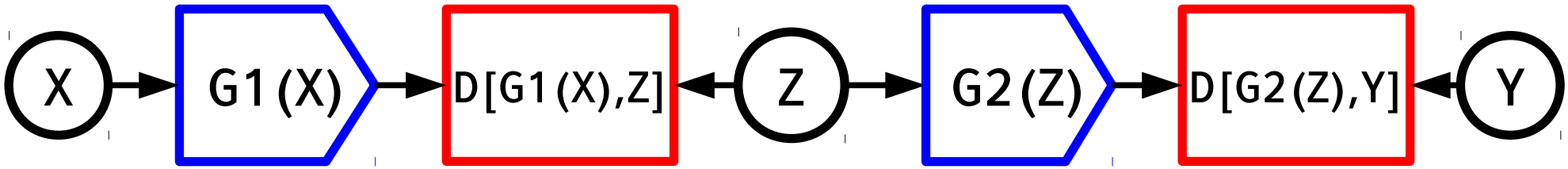


Layer Decoupling: Target Prop, PSD

- Decouple output of layer k from input of layer k+1
- Pay a (quadratic) penalty for making them different
  - [Kavuckuoglu et al. 2009]: Predictive Sparse Decomposition
  - [Mirowski & LeCun ECML 2009]: Recurrent nets



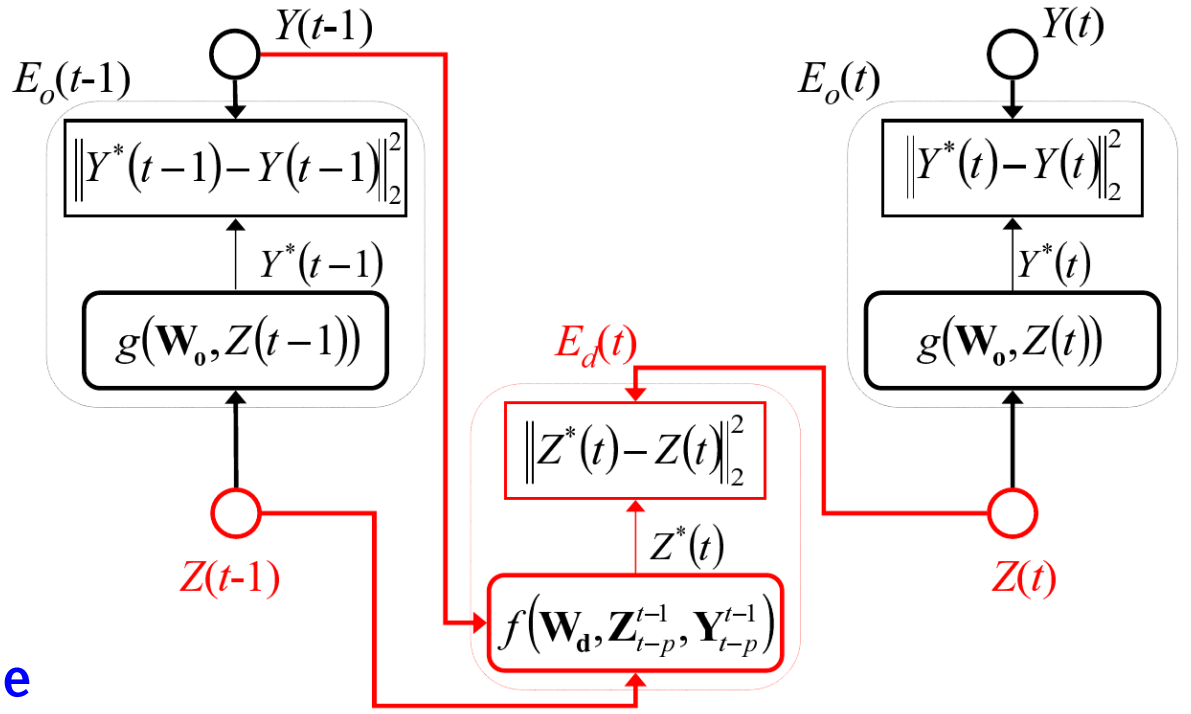
$$E(X, Y, \lambda) = D(G_2(Z), Y) + \lambda^T (G_1(X) - Z)$$

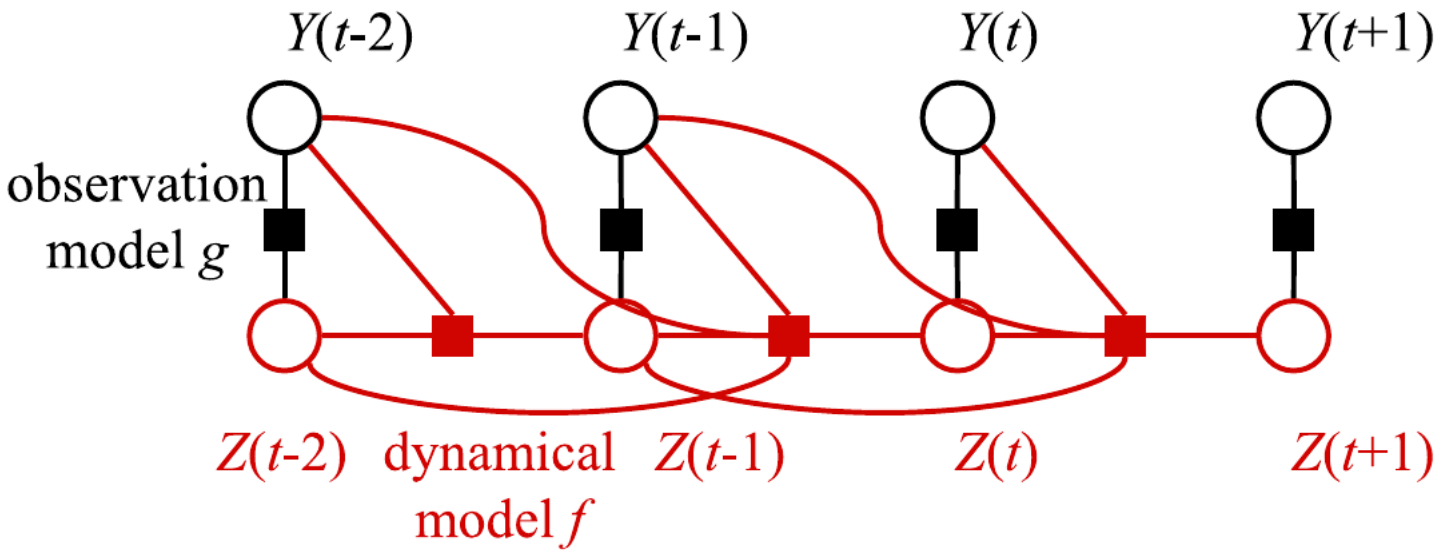


$$E(X, Y, Z) = D(G_2(Z), Y) + D(G_1(X), Z)$$

Dynamic Factor Graphs

- Target Prop for Recurrent Nets
- Deep Learning for Time-Series Prediction
  - [Mirowski & LeCun ECML 09]
  - Beats the previous SotA on the CATS time-series prediction competition dataset.
  - Temporal ConvNet with "hidden state inference"





Elastic Average SGD

Distributing SGD over multiple CPU/GPU nodes

Deep Learning withElastic Average SGD

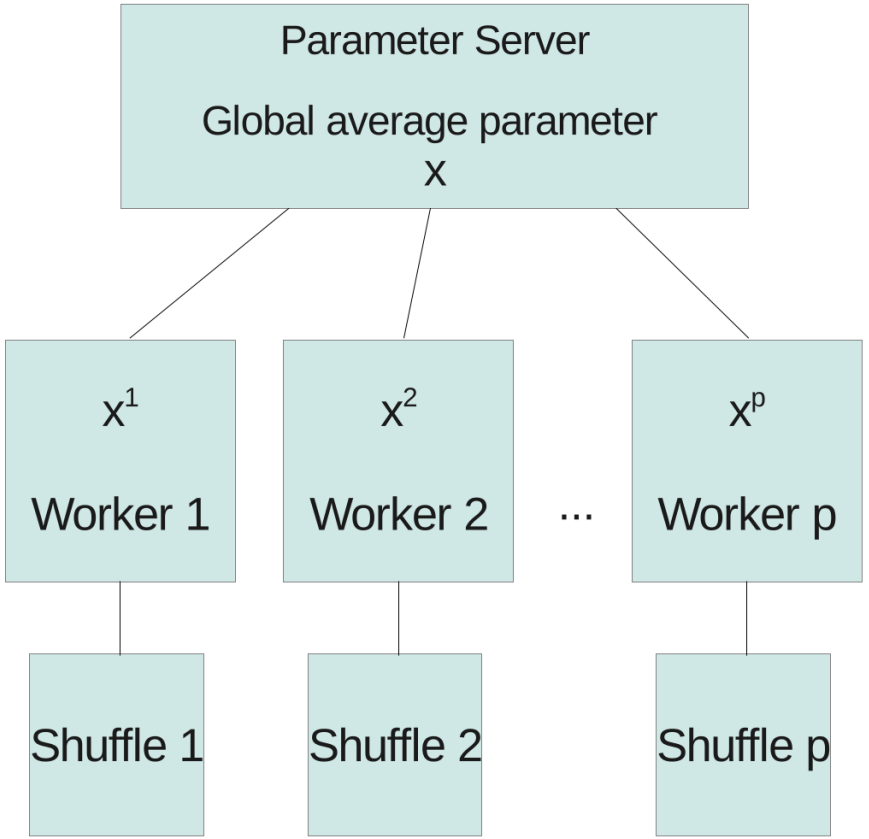
Zhang, Choromanska, LeCun, NIPS 2015 [arXiv:1412.6651]

- Expected loss

$$\min_x F(x) := \mathbb{E}[f(x, \xi)],$$

- Distributed form

$$\min_{x^1,\ldots,x^p,\tilde{x}} \sum_{i=1}^{p} \mathbb{E}[f(x^i, \xi^i)] + \frac{\rho}{2}\|x^i - \tilde{x}\|^2$$



Server updates x "slowly"
$x = x + \alpha(x^i - x)$

Worker i & server communicates with x^i and x every τ minibatch

Workers run momentum SGD asynchronously with L^2 penalty $\rho\|x^i - x\|^2$

All workers sample the full dataset

Distributing SGD over multiple CPU/GPU nodes

Deep Learning withElastic Average SGD: [Zhang, Choromanska, LeCun arXiv:1412.6651]

- Expected loss

$$\min_x F(x) := \mathbb{E}[f(x, \xi)],$$

- Distributed form

$$\min_{x^1,\ldots,x^p,\tilde{x}} \sum_{i=1}^{p} \mathbb{E}[f(x^i, \xi^i)] + \frac{\rho}{2}\|x^i - \tilde{x}\|^2$$

- Update formulas

$$\begin{aligned} x_{t+1}^i &= x_t^i - \eta(g_t^i(x_t^i) + \rho(x_t^i - \tilde{x}_t)) \\ \tilde{x}_{t+1} &= \tilde{x}_t + \eta \sum_{i=1}^{p} \rho(x_t^i - \tilde{x}_t), \end{aligned}$$

- Reparameterization:

$$\alpha = \eta\rho \text{ and } \beta = p\alpha$$

$$\begin{aligned} x_{t+1}^i &= x_t^i - \eta g_t^i(x_t^i) - \alpha(x_t^i - \tilde{x}_t) \\ \tilde{x}_{t+1} &= (1-\beta)\tilde{x}_t + \beta\left(\frac{1}{p}\sum_{i=1}^{p} x_t^i\right) \end{aligned}$$

Elastic Average SGD & Elastic Average Momentum SGD

- **Asynchronous algorithms. Sync between node every Tau updates.**
  - Every Tau steps: move workers toward center, and vice versa
  - Momentum form uses Nesterov accelerated gradient

Algorithm 1: Asynchronous EASGD: Processing by worker i and the master

Input: learning rate η, moving rate α, communication period $\tau \in \mathbb{N}$

Initialize: $\tilde{x}$ is initialized randomly, $x^i = \tilde{x}$, $t^i = 0$

Repeat

$x \leftarrow x^i$

if (τ divides t^i) **then**

a) $x^i \leftarrow x^i - \alpha(x - \tilde{x})$

b) $\tilde{x} \leftarrow \tilde{x} + \alpha(x - \tilde{x})$

end

$x^i \leftarrow x^i - \eta g^i_{t^i}(x)$

$t^i \leftarrow t^i + 1$

Until forever

Algorithm 2: Asynchronous EAMSGD: Processing by worker i and the master

Input: learning rate η, moving rate α, communication period $\tau \in \mathbb{N}$, momentum term δ

Initialize: $\tilde{x}$ is initialized randomly, $x^i = \tilde{x}$, $v^i = 0$, $t^i = 0$

Repeat

$x \leftarrow x^i$

if (τ divides t^i) **then**

a) $x^i \leftarrow x^i - \alpha(x - \tilde{x})$

b) $\tilde{x} \leftarrow \tilde{x} + \alpha(x - \tilde{x})$

end

$v^i \leftarrow \delta v^i - \eta g^i_{t^i}(x + \delta v^i)$

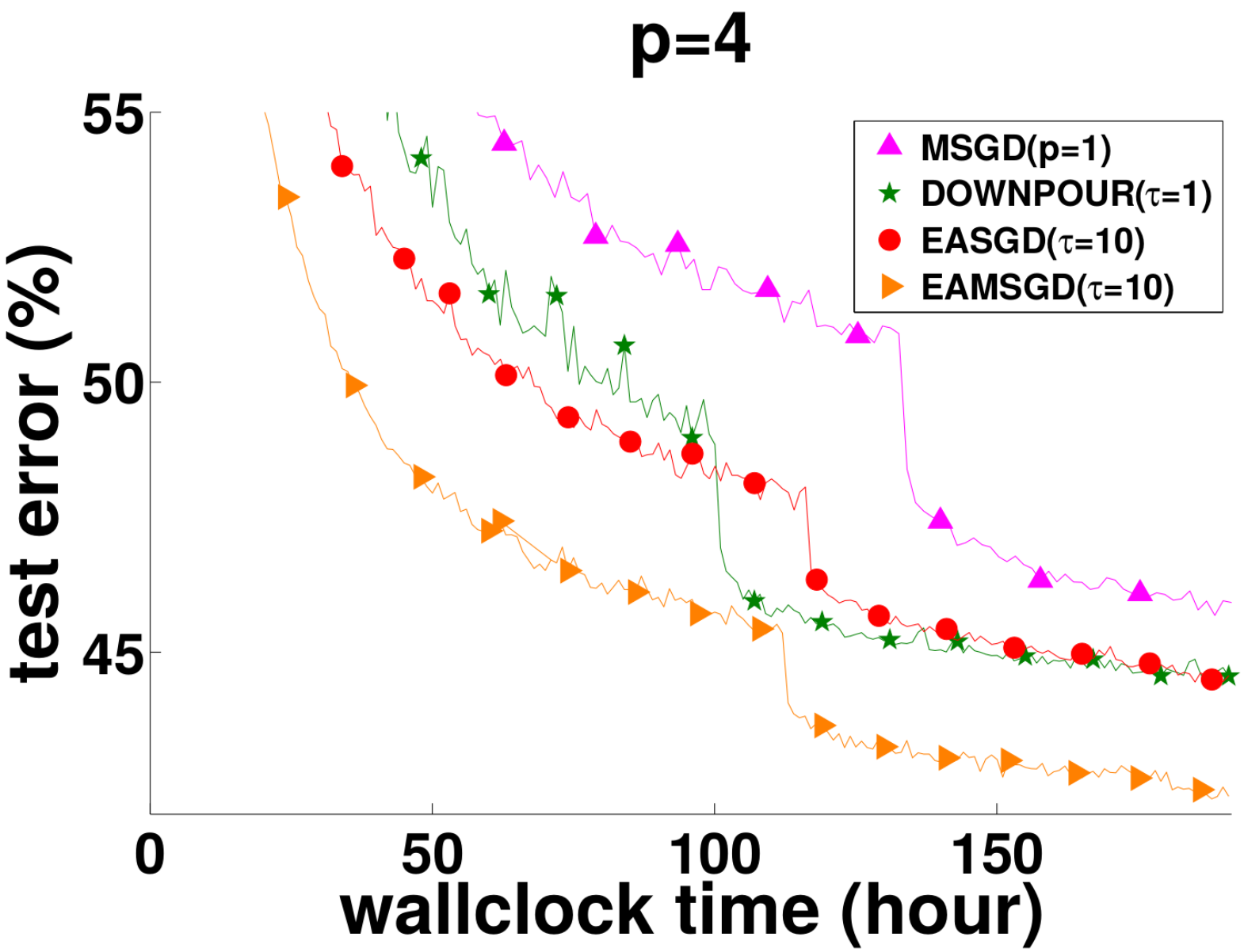
$x^i \leftarrow x^i + v^i$

$t^i \leftarrow t^i + 1$

Until forever

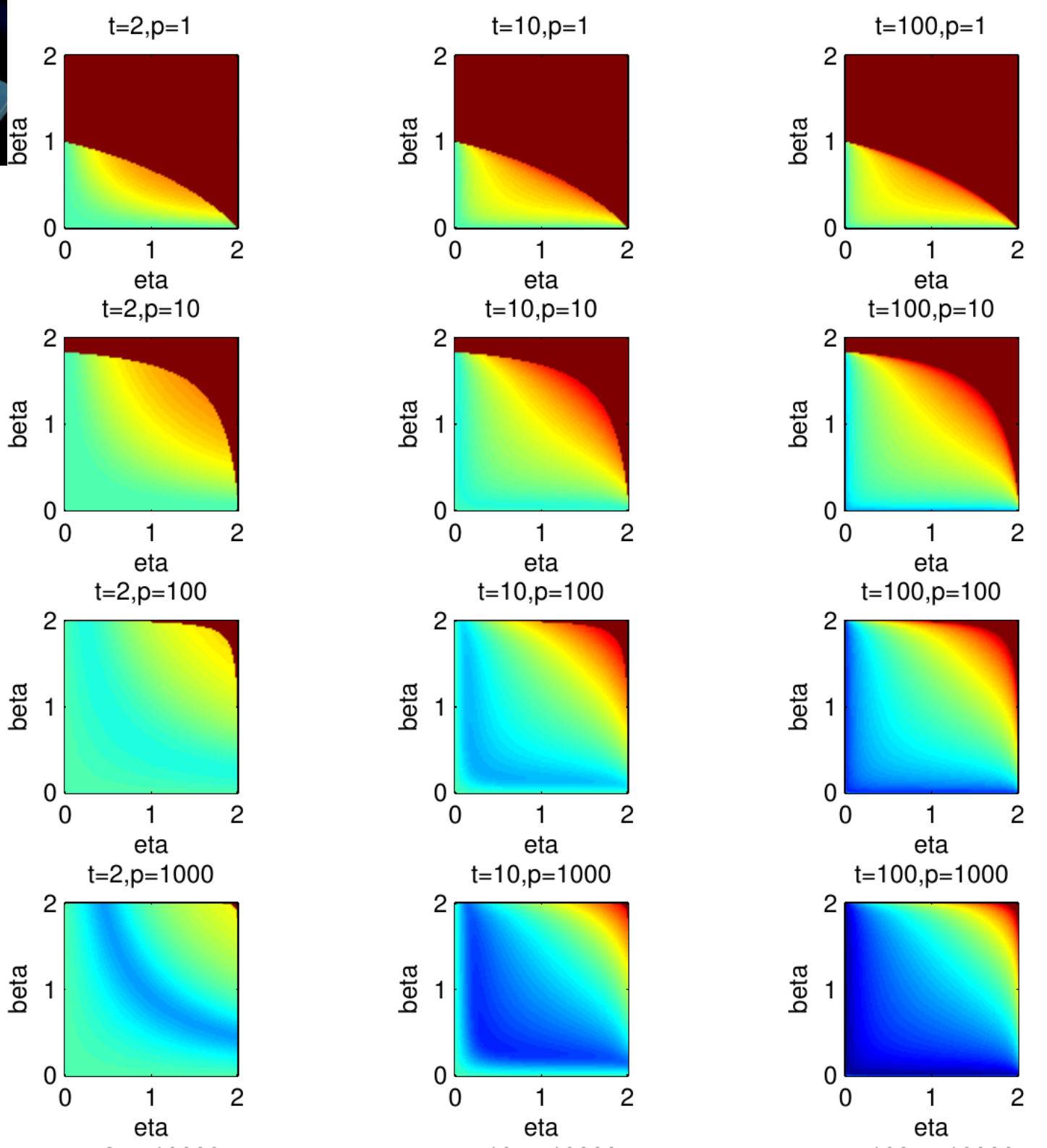
Elastic Average SGD: Results on ImageNet

- Compared with Downpour and Momentum SGD.



Analysis Quadratic Case

- P: number of nodes
- T: period of sync
- Eta: step size
- Beta= Alpha*p
  - averaging rate



Downpour algorithm: send gradient, receive parameter vector

- Asynchronous algorithm. Sync between node every Tau updates.
  - Every Tau steps:
    - node sends accumulated gradient to server
    - Server sends updated parameter vector to node.
  - Momentum form uses Nesterov accelerated gradient

Algorithm 3: DOWNPOUR: Processing by worker i and the master

Input: learning rate η, communication period $\tau \in \mathbb{N}$
Initialize: $\tilde{x}$ is initialized randomly, $x^i = \tilde{x}$, $v^i = 0$, $t^i = 0$

Repeat
  if (τ divides t^i) **then**
    $\tilde{x} \leftarrow \tilde{x} + v^i$
    $x^i \leftarrow \tilde{x}$
    $v^i \leftarrow 0$
  end
  $x^i \leftarrow x^i - \eta g^i_{t^i}(x^i)$
  $v^i \leftarrow v^i - \eta g^i_{t^i}(x^i)$
  $t^i \leftarrow t^i + 1$
Until forever

Like ADMM without the constraint term

- But ADMM is unstable under a round robin scheme for synchronization

$$\max_{\lambda^1,\ldots,\lambda^p} \min_{x^1,\ldots,x^p,\tilde{x}} \sum_{i=1}^{p} F(x^i) - \lambda^i(x^i - \tilde{x}) + \frac{\rho}{2}\|x^i - \tilde{x}\|^2$$

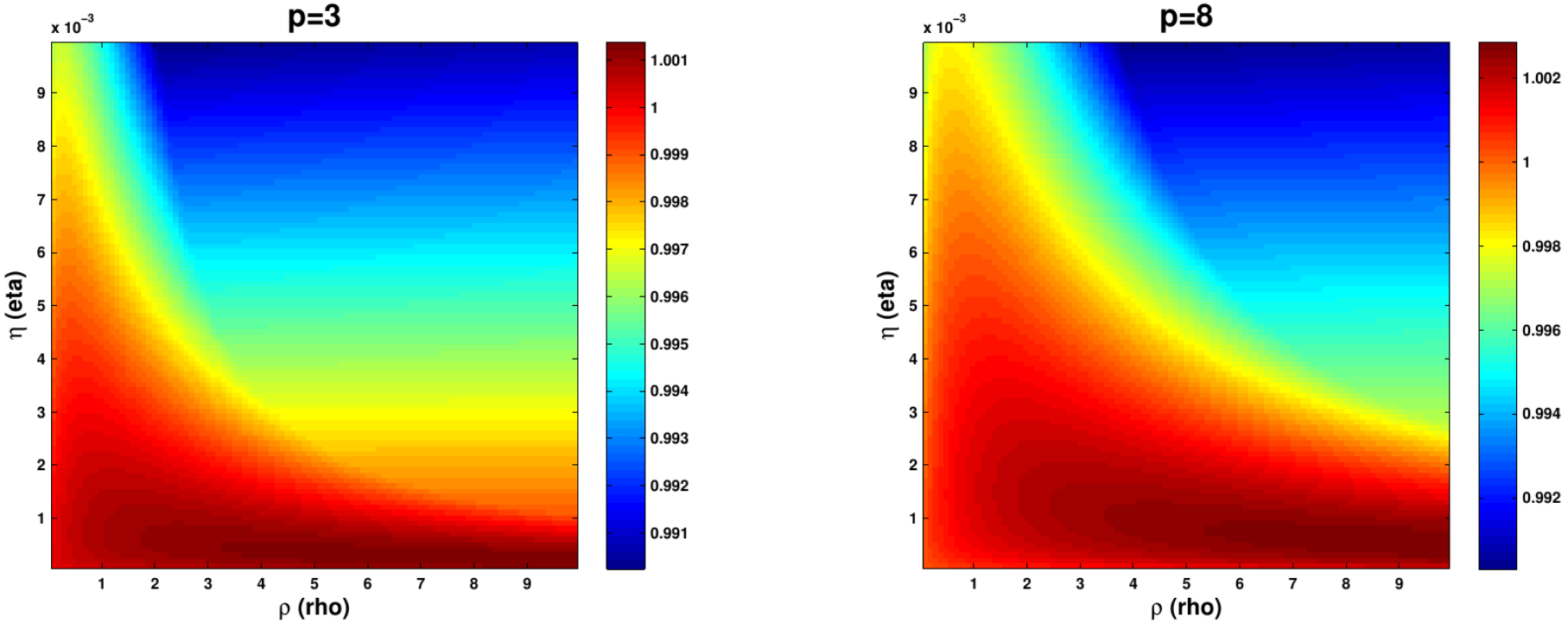
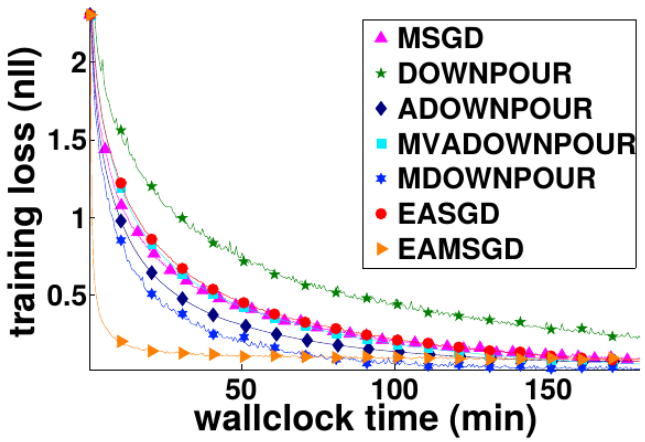
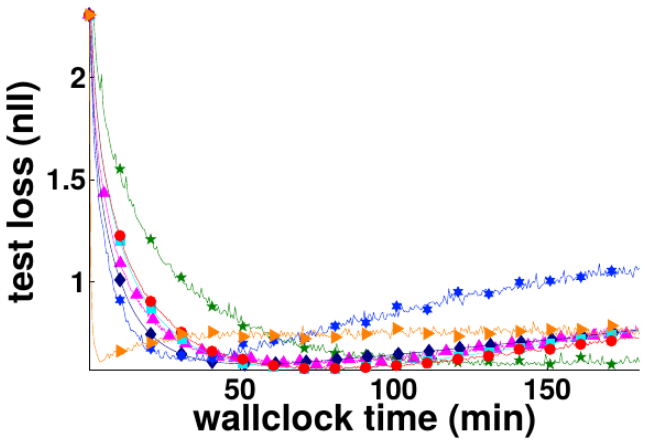


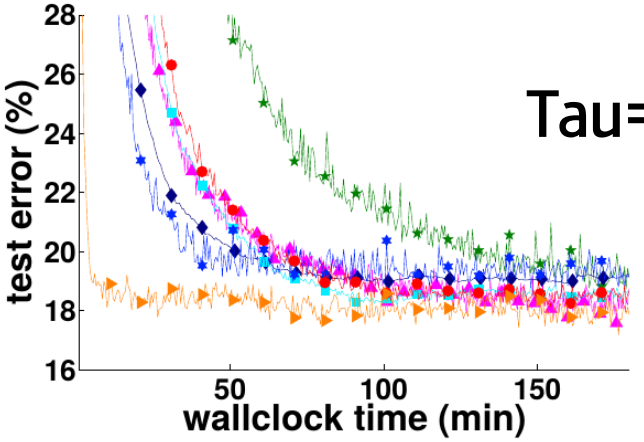
Figure 1: The largest absolute eigenvalue of the linear map $\mathcal{F} = F_3^p \circ F_2^p \circ F_1^p \circ \ldots \circ F_3^1 \circ F_2^1 \circ F_1^1$ as a function of $\eta \in (0, 10^{-2})$ and $\rho \in (0, 10)$ when $p = 3$ and $p = 8$. To simulate the chaotic behavior of the *ADMM* algorithm, one may pick $\eta = 0.001$ and $\rho = 2.5$ and initialize the state s_0 either randomly or with $\lambda_0^1 = 0, x_0^1 = 1000, \lambda_0^2 = 0, x_0^2 = 1000, \lambda_0^3 = 0, x_0^3 = 1000, \tilde{x}_0 = 1000$. *Figure should be read in color.*

Results on CIFAR-10 dataset , 7-layer ConvNet, 4 nodes

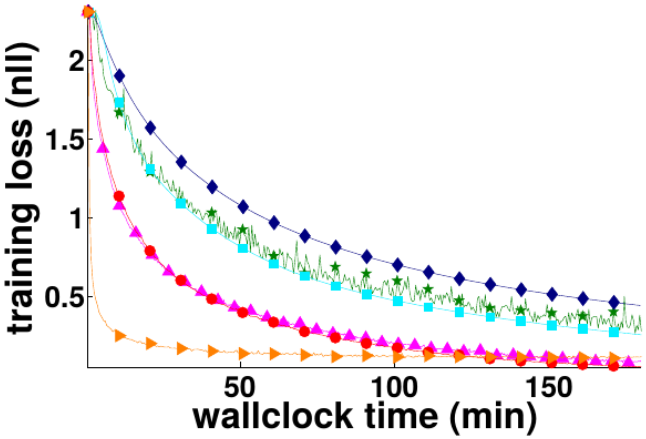
Y LeCun

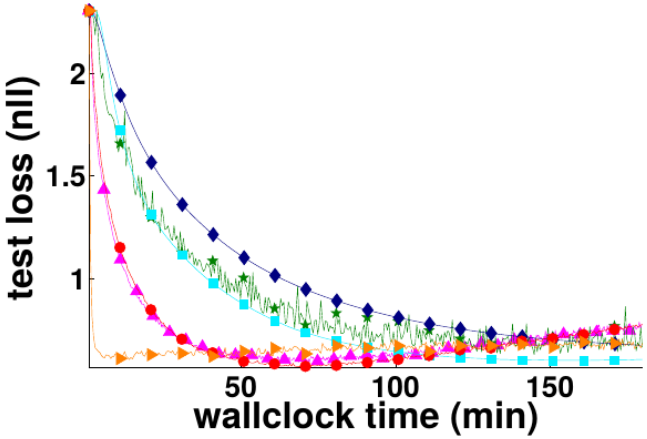


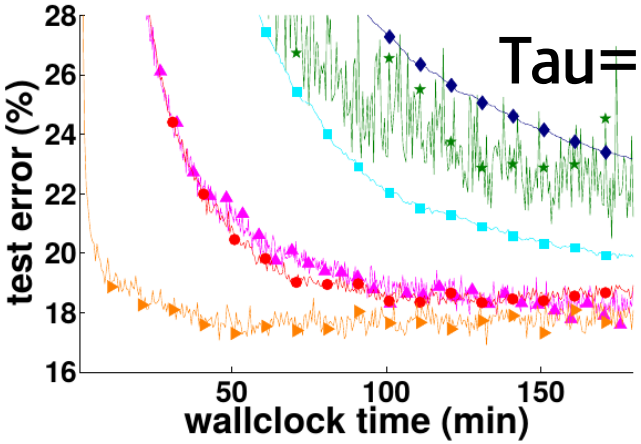




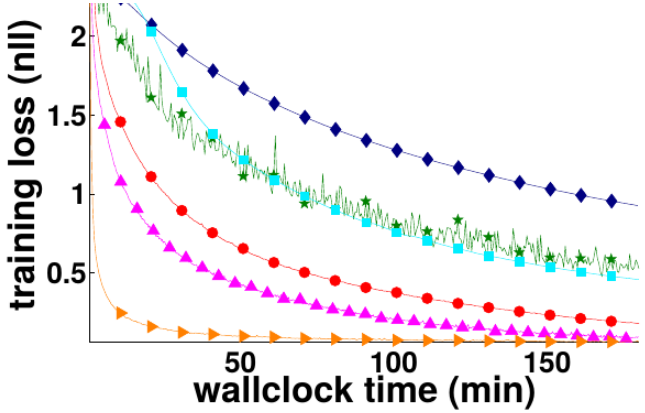
Tau=1

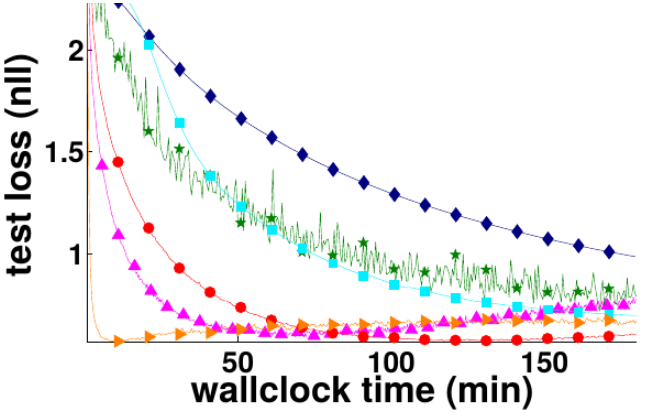


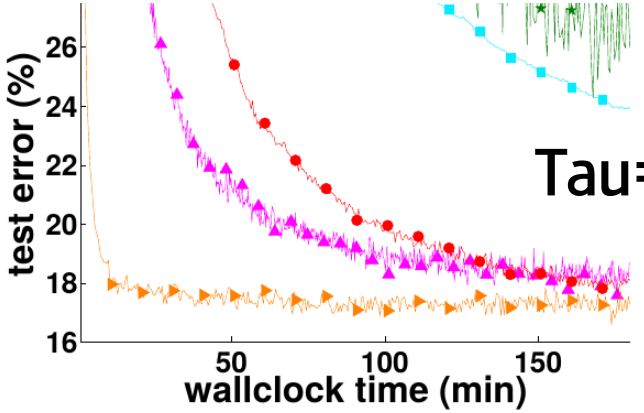




Tau=16



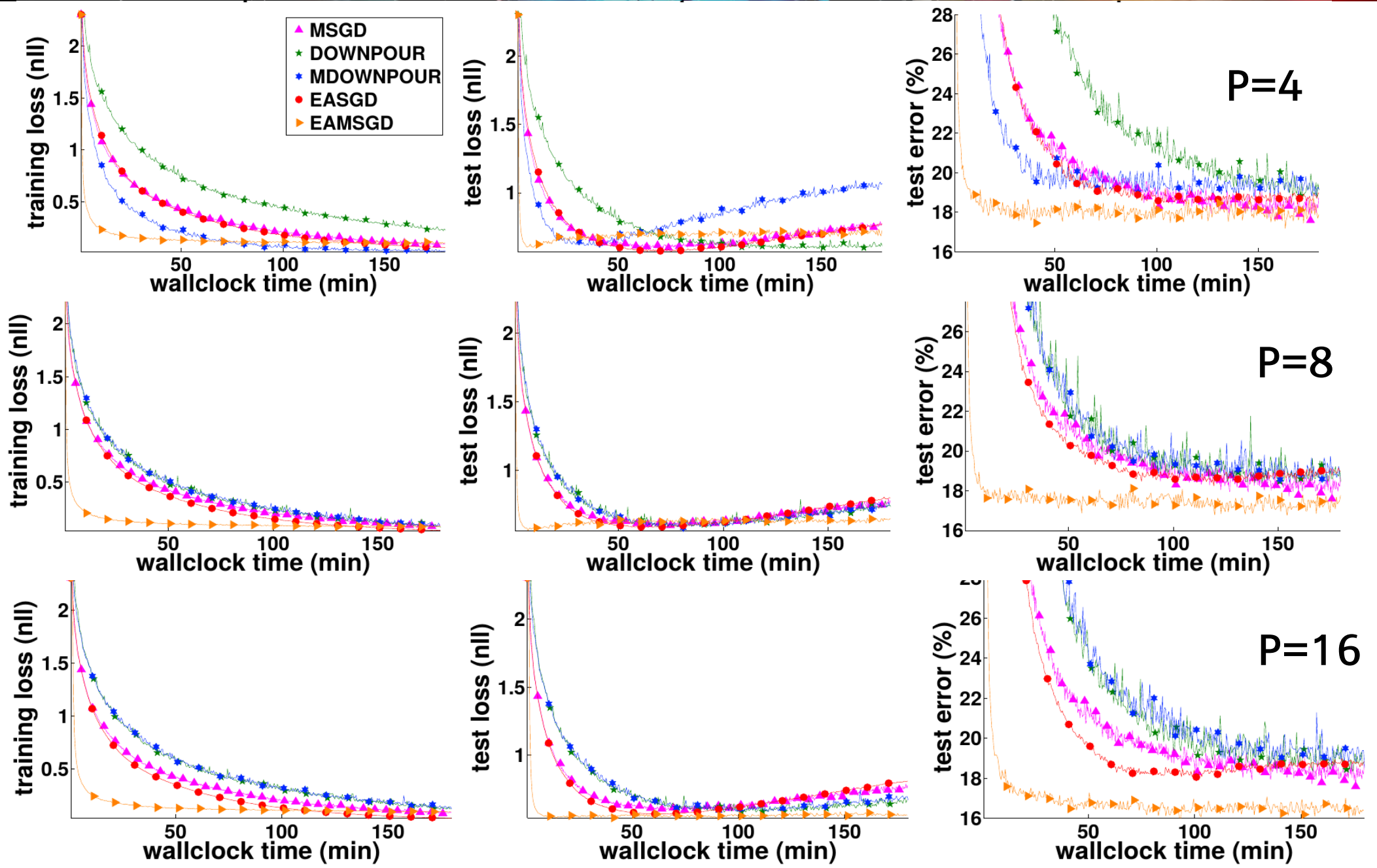




Tau=64

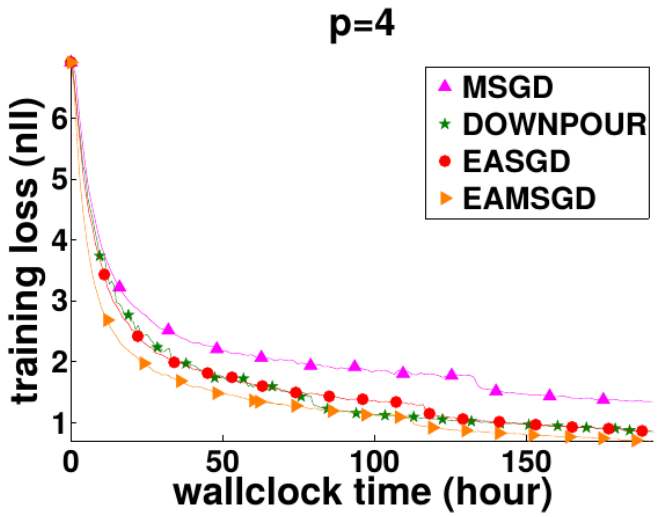
Results: CIFAR-10, 7-layer ConvNet, Tau=10 (Tau=1 for Downpour)

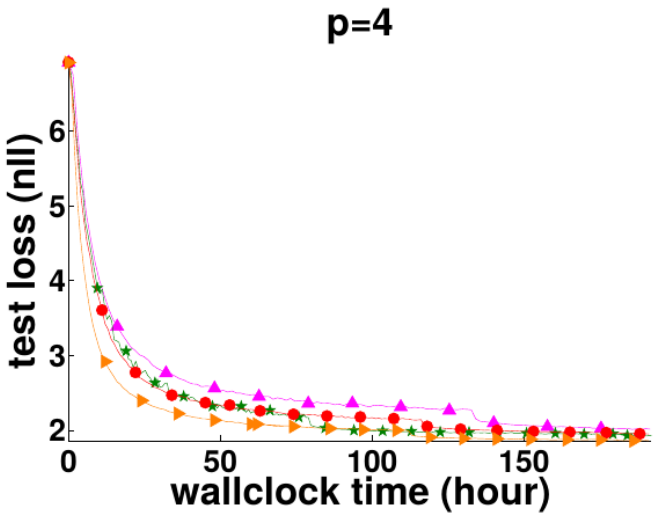
Y LeCun

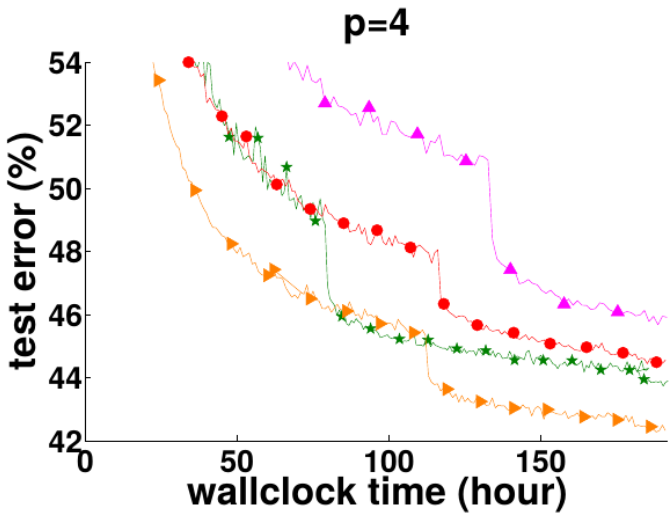


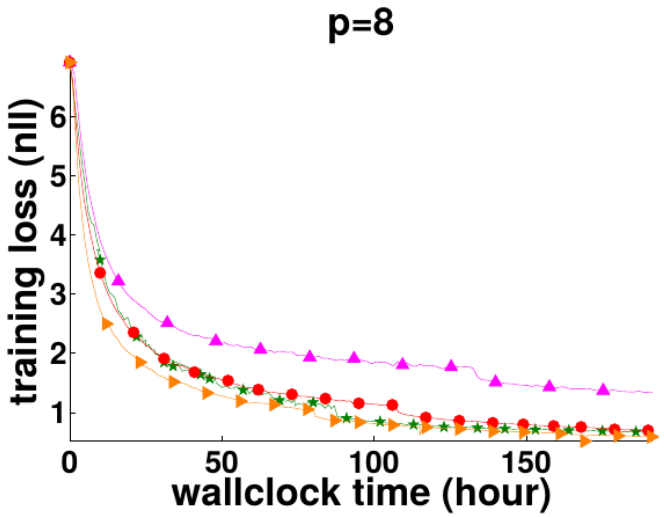
Results: ImageNet, ConvNet, Tau=10 (Tau=1 for Downpour)

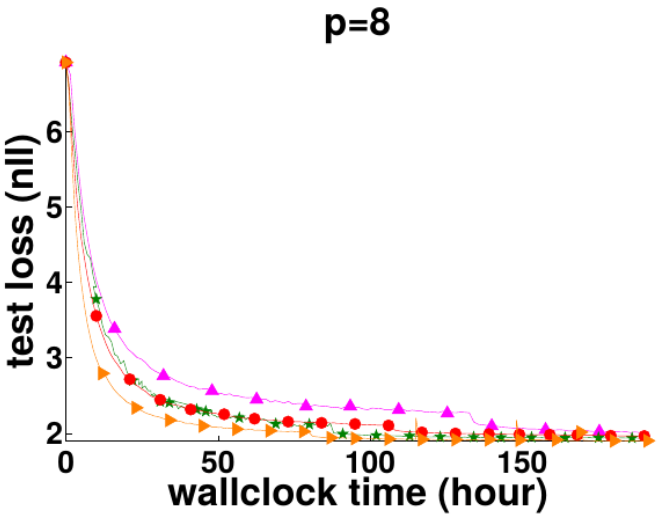
Y LeCun

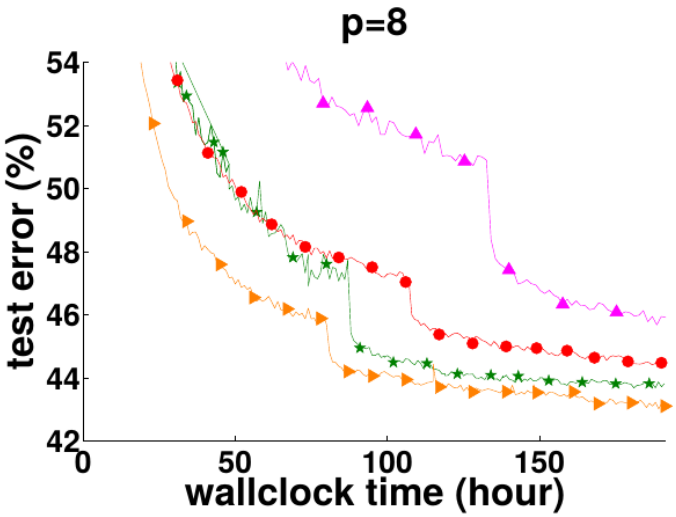












Results: Performance comparison on CIFAR-10 and ImageNet

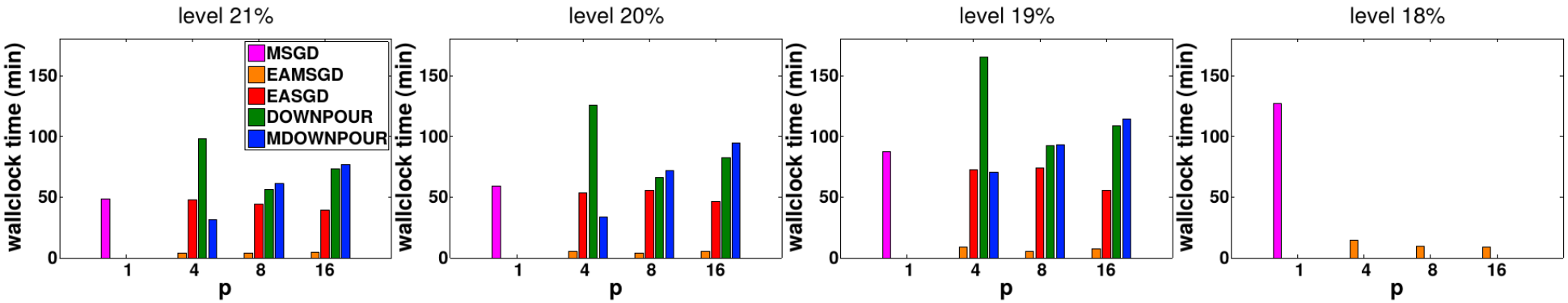


Figure 10: The wall clock time needed to achieve the same level of the test error thr as a function of the number of local workers p on the *CIFAR* dataset. From left to right: $thr = \{21\%, 20\%, 19\%, 18\%\}$. Missing bars denote that the method never achieved specified level of test error.

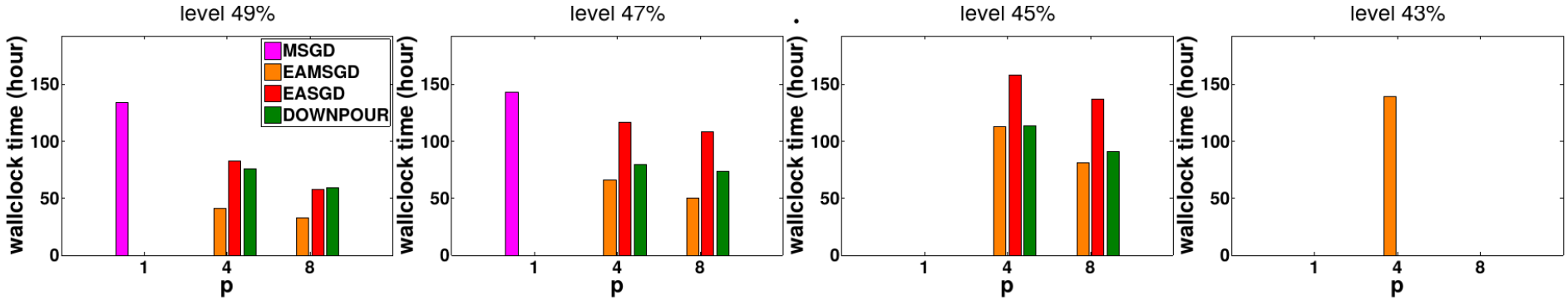


Figure 11: The wall clock time needed to achieve the same level of the test error thr as a function of the number of local workers p on the *ImageNet* dataset. From left to right: $thr = \{49\%, 47\%, 45\%, 43\%\}$. Missing bars denote that the method never achieved specified level of test error.